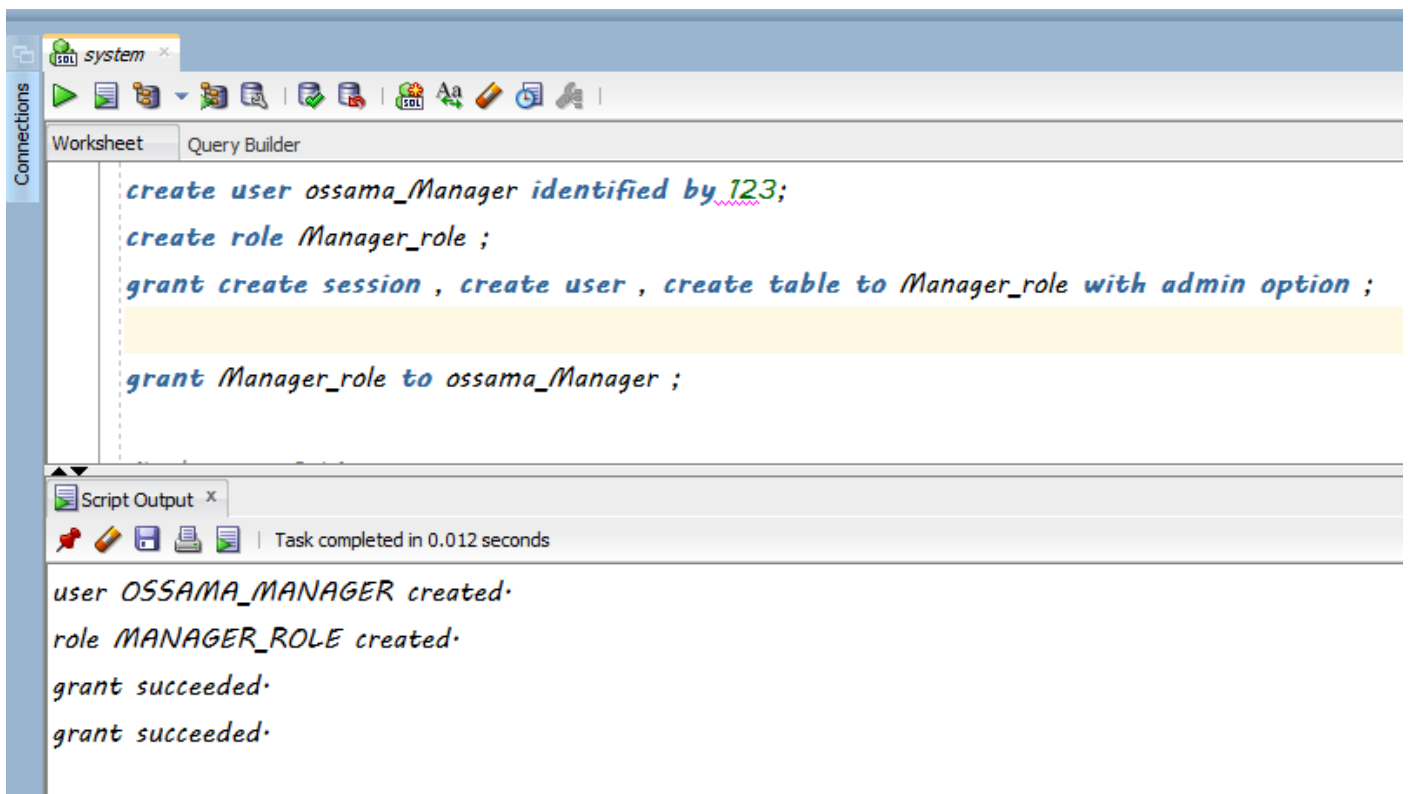


## ORACLE TASK:

An organization has a "Department" table and an "Employees" table in Oracle. The "Department" table contains information about different departments in the organization, and the "Employees" table contains information about the employees, including the department they belong to. The department table contains ID as a primary key and department name. The departments are HR, IT, and finance. The Employee table contains ID as a primary key and name, salary, and department ID as a foreign key.

- a) Create a Manager User and grant them a role of privileges to create two users. Let User 1 create the Employee and the Department table. Let User 2 insert 5 rows of employees.

system create manager and grant privileges to create session and  
create user and create table with admin option to grant it to user1



The screenshot shows the Oracle SQL Developer interface. The top toolbar includes icons for running queries, saving, and other database operations. The main window is titled 'Worksheet' and 'Query Builder'. The SQL script being executed is as follows:

```
create user ossama_Manager identified by 123;
create role Manager_role ;
grant create session , create user , create table to Manager_role with admin option ;

grant Manager_role to ossama_Manager ;
```

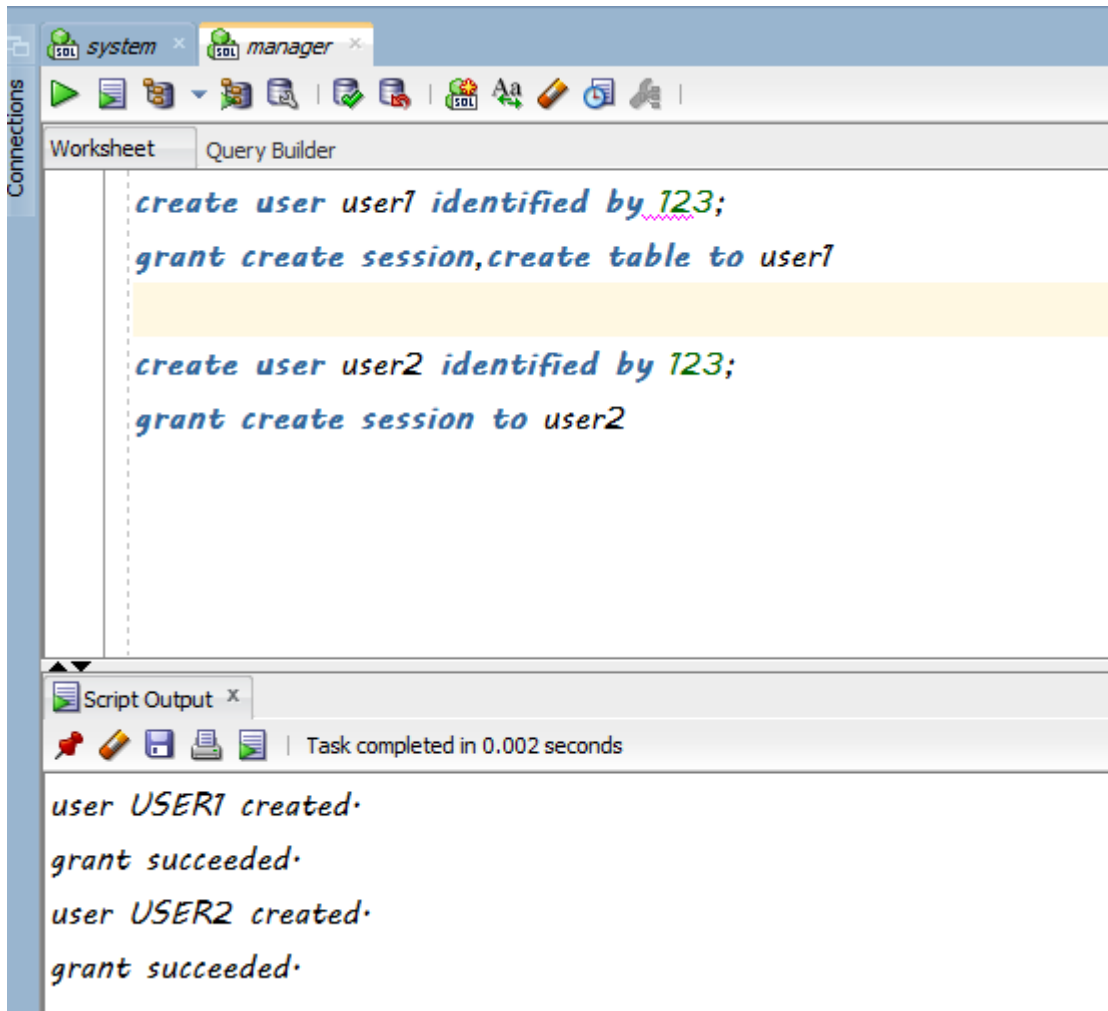
Below the script, the 'Script Output' window shows the results of the execution:

```
user OSSAMA_MANAGER created.
role MANAGER_ROLE created.
grant succeeded.
grant succeeded.
```

The output also indicates that the task was completed in 0.012 seconds.

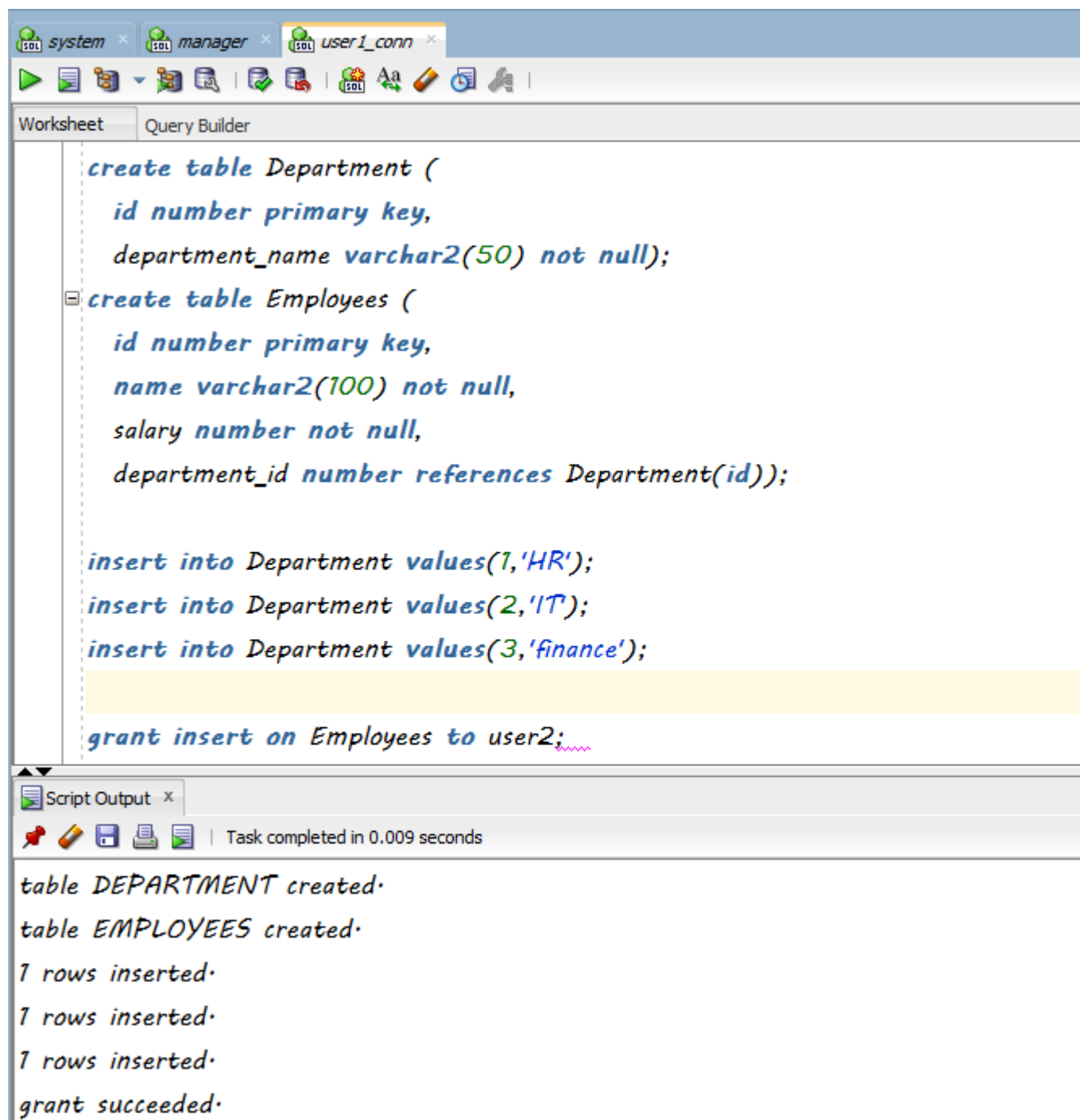
```
/* alter user1 */  
alter user user1 quota 100M on system;
```

Alter user1 to can create tables



Manager create two users and grant create session to them

And grant create table to only user1



The screenshot shows the SQL Developer interface with three tabs: 'system', 'manager', and 'user1\_conn'. The 'Query Builder' tab is active, displaying a SQL script. The script creates two tables, 'Department' and 'Employees', with their respective columns and constraints. It then inserts three rows into the 'Department' table and grants insert privileges on the 'Employees' table to 'user2'. The 'Script Output' window at the bottom shows the execution results, confirming the successful creation of tables, insertion of rows, and granting of privileges.

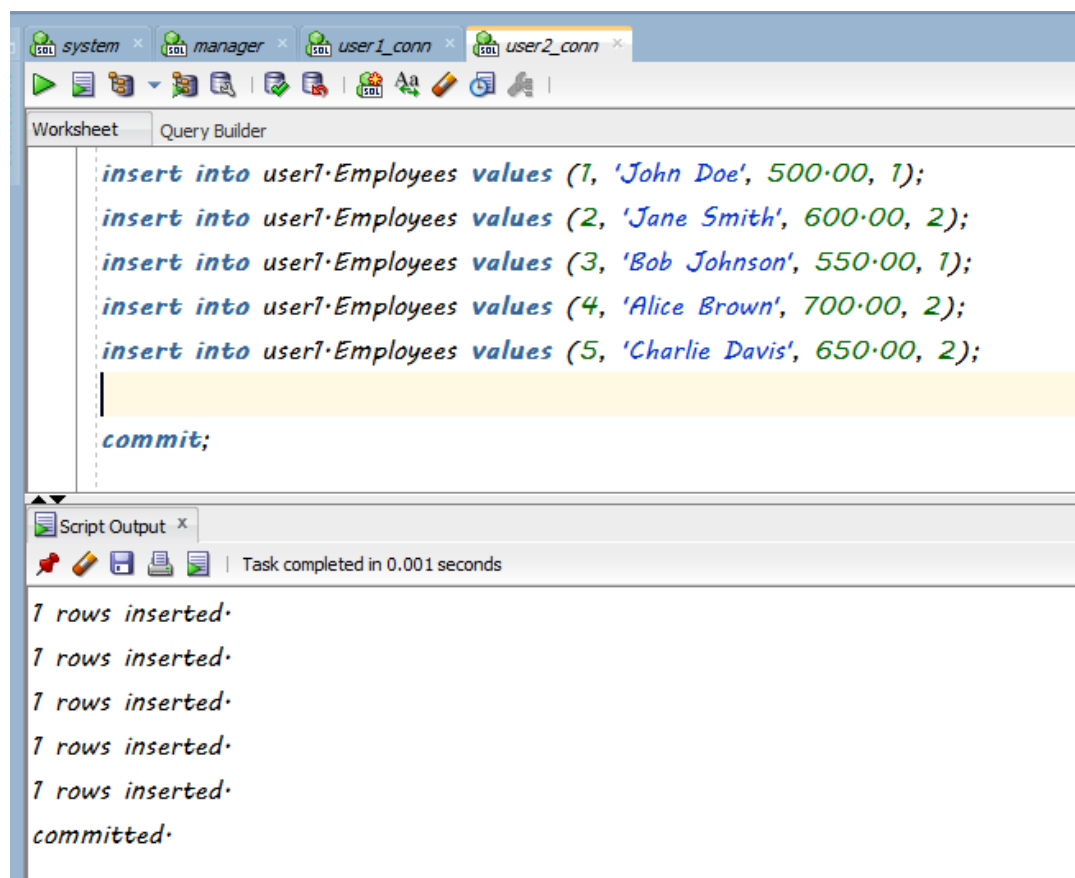
```
create table Department (  
    id number primary key,  
    department_name varchar2(50) not null);  
  
create table Employees (  
    id number primary key,  
    name varchar2(100) not null,  
    salary number not null,  
    department_id number references Department(id));  
  
insert into Department values(1,'HR');  
insert into Department values(2,'IT');  
insert into Department values(3,'finance');  
  
grant insert on Employees to user2;
```

Script Output x

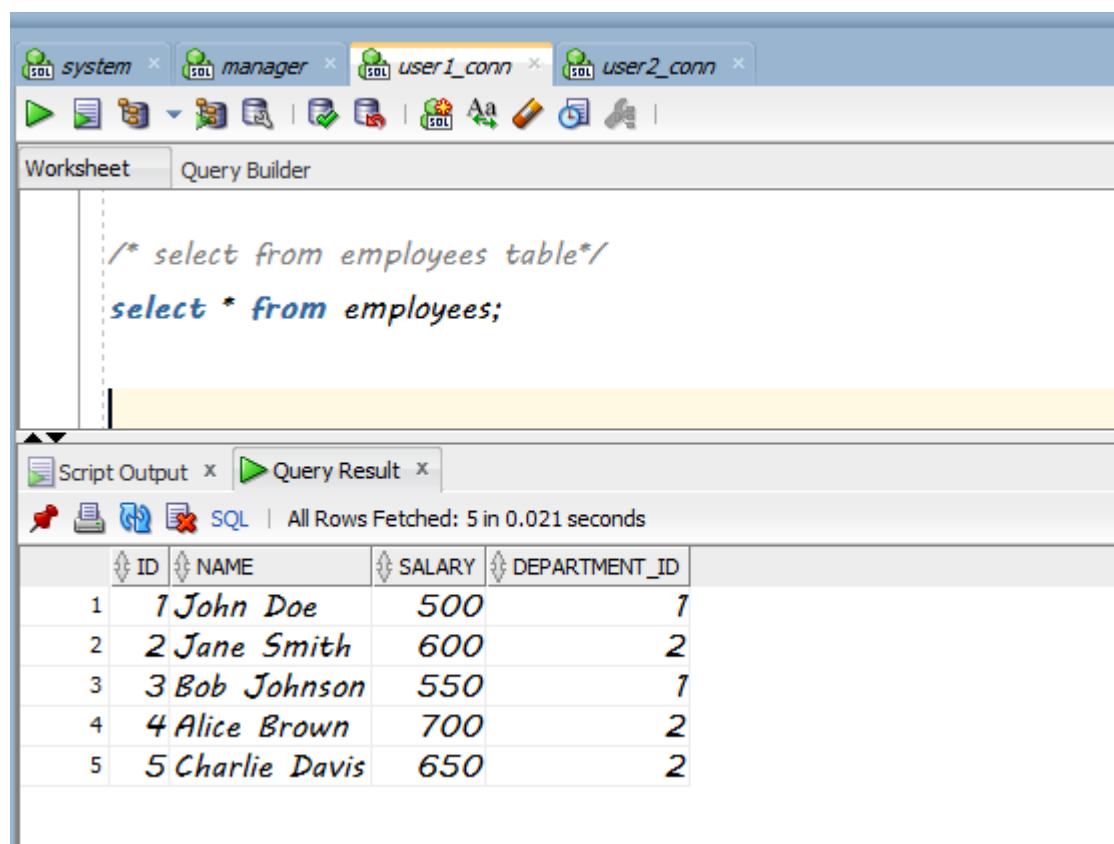
Task completed in 0.009 seconds

```
table DEPARTMENT created.  
table EMPLOYEES created.  
1 rows inserted.  
1 rows inserted.  
1 rows inserted.  
grant succeeded.
```

User1 create two table department and employees and insert there rows in table department



User2 insert 5 rows in table department and commit

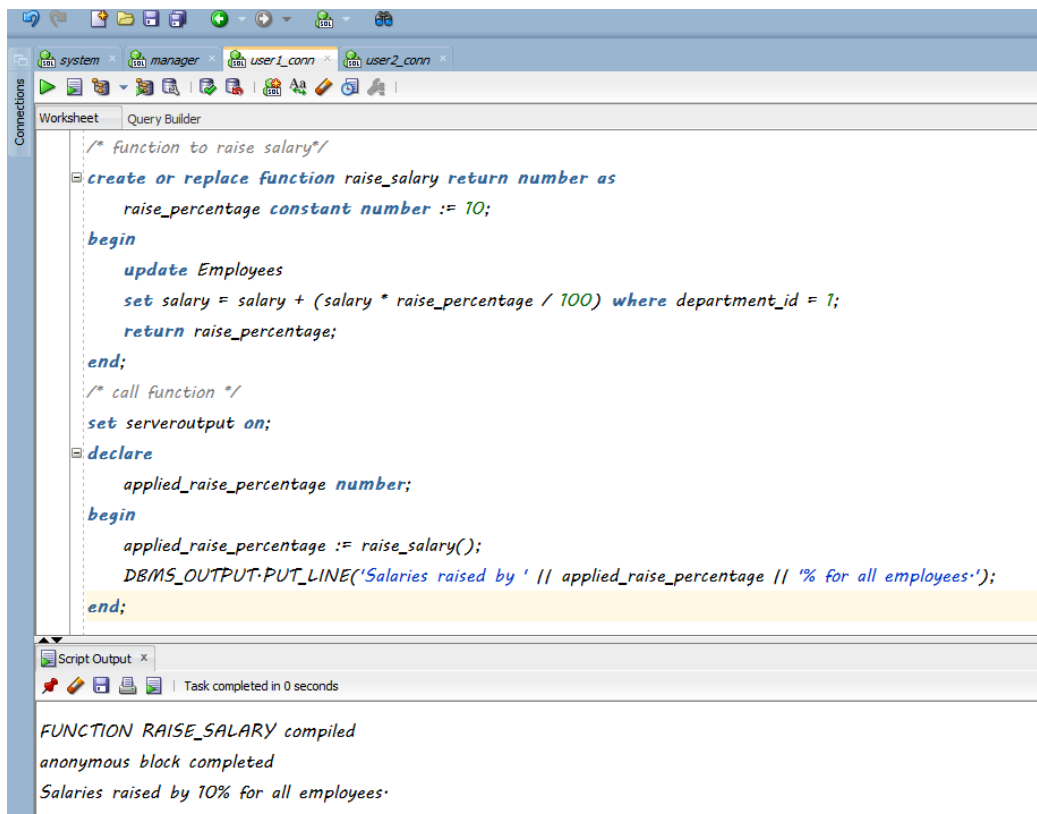


User1 select all rows in employees after user2 commit

- b)** Demonstrate generating a blocker-waiting situation using two transactions by user 1 and user 2. The Transaction is calling a function that raises the rate of salary by 10% for department 1.

```
/* grant create function to user2*/
grant create procedure to user1;
```

System grant user1 to create function or procedure



The screenshot shows the SQL Developer interface with a query window containing the following PL/SQL code:

```
/* function to raise salary*/
create or replace function raise_salary return number as
    raise_percentage constant number := 10;
begin
    update Employees
    set salary = salary + (salary * raise_percentage / 100) where department_id = 1;
    return raise_percentage;
end;
/* call function */
set serveroutput on;
declare
    applied_raise_percentage number;
begin
    applied_raise_percentage := raise_salary();
    DBMS_OUTPUT.PUT_LINE('Salaries raised by ' || applied_raise_percentage || '% for all employees');
end;
```

Below the code window, the 'Script Output' window shows the following messages:

```
FUNCTION RAISE_SALARY compiled
anonymous block completed
Salaries raised by 10% for all employees.
```

User1 create function to raise salary for department 1 by 10% and execute function

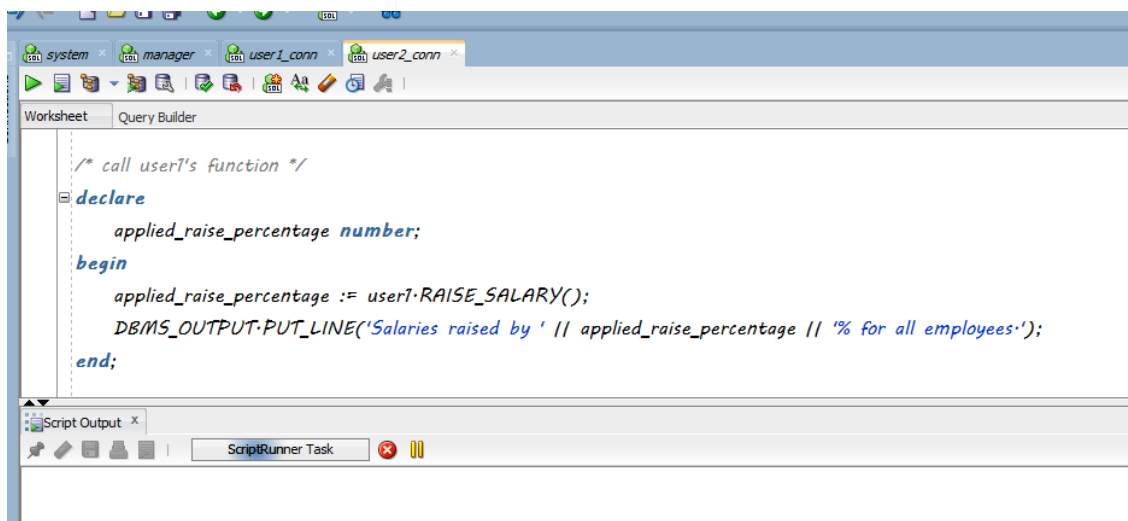
```

/* grant execute function to user2*/
grant execute on raise_salary to user2;

```

User1 grant execute function raise salary to user2

- c) Identify the sessions in the situation using SID and serial# for both blocker and waiting sessions.



User2 execute raise\_salary and Enters a waiting state

The screenshot shows the SQL Developer interface with a query window open. The query is as follows:

```

/* select SID and sreal# for blocker and waiting usres*/
select
  blocking_session.sid as blocking_sid,
  blocking_session.SERIAL# as blocking_serial,
  waiting_session.sid as waiting_sid,
  waiting_session.SERIAL# as waiting_serial
from
  v$session blocking_session,
  v$session waiting_session
where
  blocking_session.sid = waiting_session.BLOCKING_SESSION;

```

The Query Result pane shows the following data:

	BLOCKING_SID	BLOCKING_SERIAL	WAITING_SID	WAITING_SERIAL
1	173	453	135	141

System select blocker an waiting info

d) Demonstrate a deadlock scenario and display the expected result.

The screenshot shows the SQL Developer interface with a query window open. The query is as follows:

```

/* update employee 1 */
update Employees set salary = salary + salary * 0.1 where Employees.ID = 1;

```

The Script Output pane shows the following message:

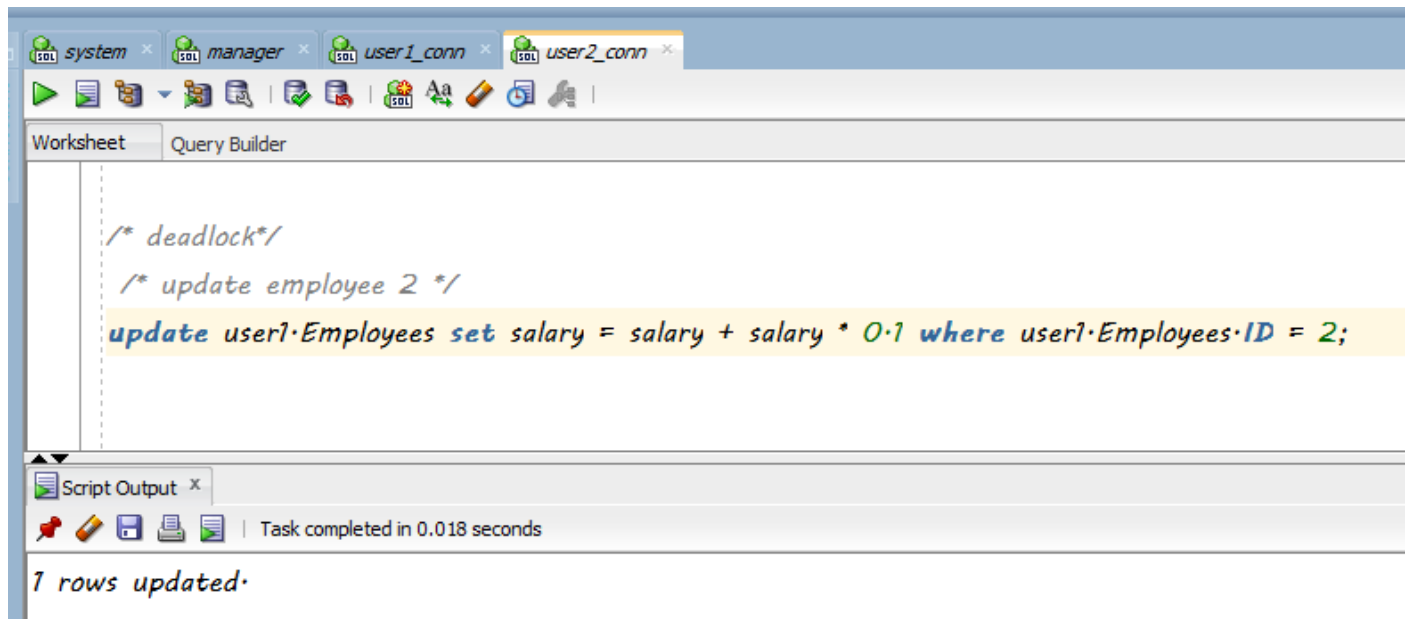
Task completed in 0.002 seconds

1 rows updated.

User1 update employee have id = 1 in table employees  
But don't make commit

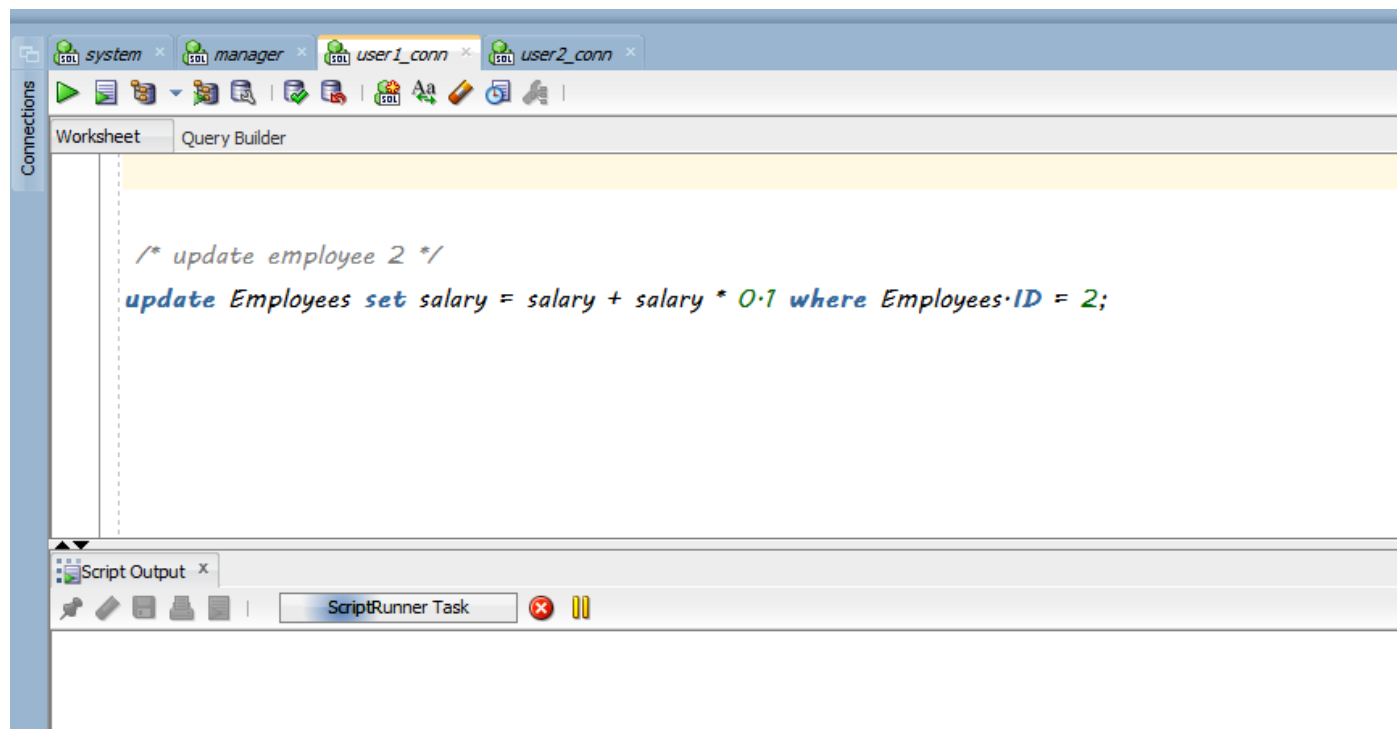
```
/* deadlock*/
grant update on Employees to user2;
```

User1 grant update on employees table employees  
But granting done before user1 make update

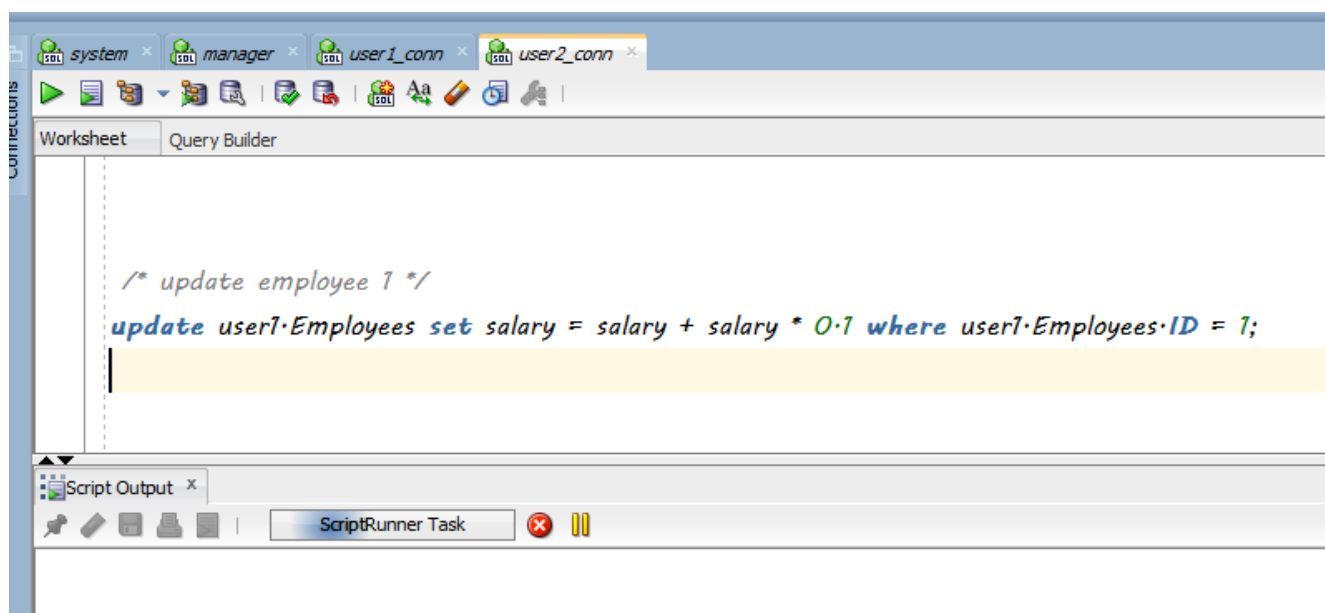


User2 update employee have id = 2 in table employees  
But don't make commit





user1 try to update employee have id = 2 and Enters a waiting state



user2 try to update employee have id = 1 and Enters a waiting state

The screenshot shows the Oracle SQL Developer interface. At the top, there are tabs for 'system', 'manager', 'user1\_conn', and 'user2\_conn'. Below the tabs is a toolbar with various icons. The main window is titled 'Worksheet' and contains the following SQL code:

```
update Employees set salary = salary + salary * 0.1 where Employees.ID = 1;

/* update employee 2 */
update Employees set salary = salary + salary * 0.1 where Employees.ID = 2;
```

Below the code editor is a 'Script Output' window. It shows the following text:

Task completed in 39.189 seconds

Error starting at line : 54 in command -

```
update Employees set salary = salary + salary * 0.1 where Employees.ID = 2
```

Error report -

SQL Error: ORA-00060: **deadlock** detected while waiting for resource

00060. 00000 - "deadlock detected while waiting for resource"

\*Cause: Transactions deadlocked one another while waiting for resources.

\*Action: Look at the trace file to see the transactions and resources involved. Retry if necessary.

Each user in waiting state that is called deadlock

e) Perform the following functions

- i. Create a function that calculates the average salary for any department

The screenshot shows a SQL IDE window with multiple tabs: 'system', 'manager', 'user1\_conn', 'user2\_conn', and 'EMPLOYEES'. The 'Query Builder' tab is active, displaying the following SQL code:

```

/*Create a function that calculates the average salary for any department */
create or replace function avg_salary(dept_id number)
return number is avg_sal number;

begin
    select avg(salary)
    into avg_sal
    from employees
    where department_id = dept_id;
    return avg_sal;
end;

/* executer avg_salary function */
set serveroutput on;

declare avg_sal number;
begin
    avg_sal := avg_salary(1);
    DBMS_OUTPUT.PUT_LINE('Average Salary is: ' || avg_sal);
end;
  
```

Below the code editor, the 'Script Output' window shows the execution results:

```

Task completed in 0.002 seconds

anonymous block completed
Average Salary is: 525
  
```

- ii. Create a function that calculates the Total Salary in a Department.

The screenshot shows the SQL Developer interface with the following components:

- Top Bar:** Contains tabs for 'system', 'manager', 'user1\_conn', 'user2\_conn', and 'EMPLOYEES'. Below the tabs is a toolbar with various icons for execution, saving, and editing.
- Worksheet Tab:** The active tab is 'Query Builder'.
- Code Editor:** Contains the following PL/SQL code:
 

```

/*Create a function that calculates the Total Salary in a Department*/
create or replace function total_salary(dept_id number)
return number is total_sal number;
begin
  select sum(salary)
  into total_sal
  from employees
  where department_id = dept_id;
  return total_sal;
end;

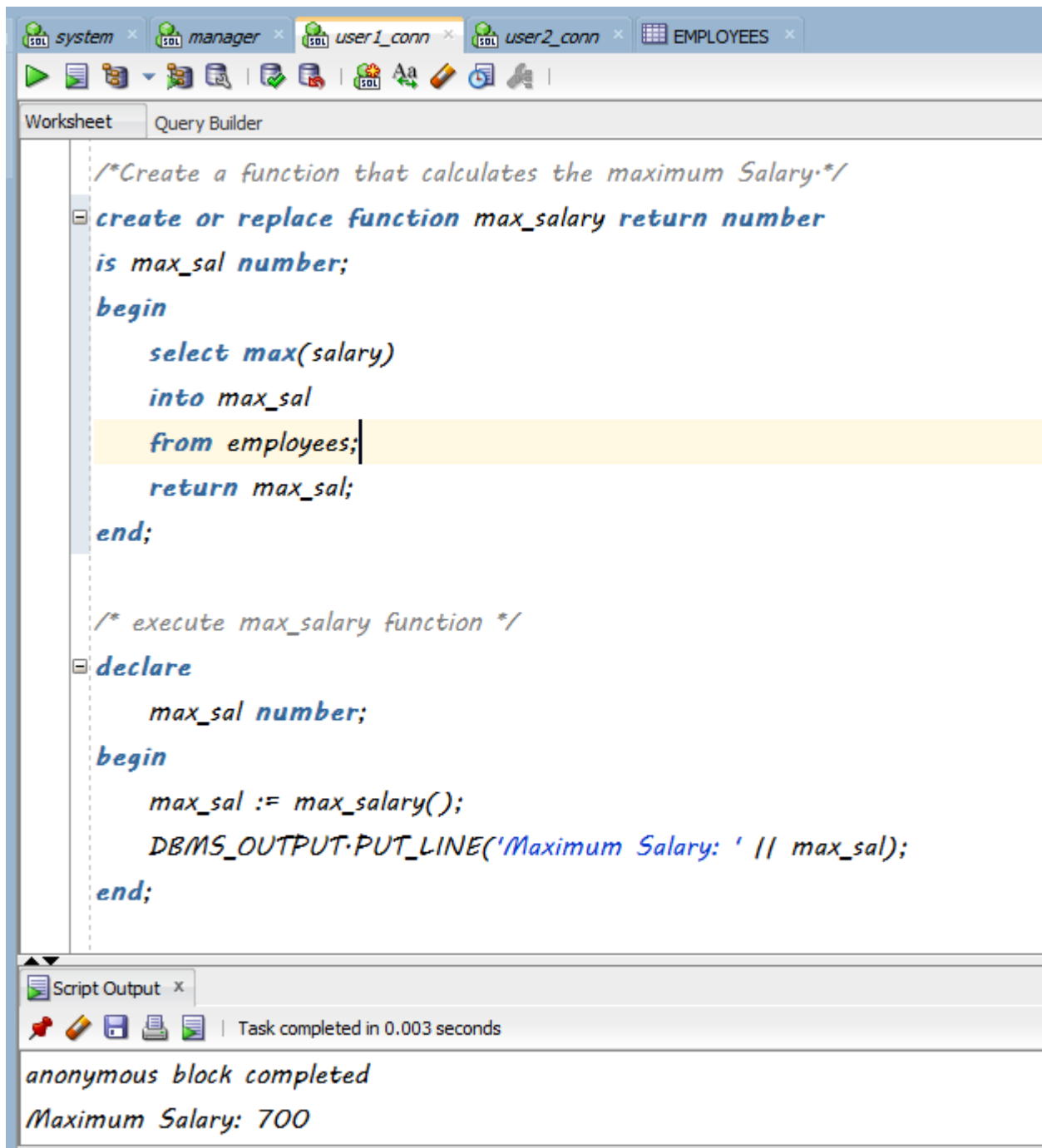
/* execute total salary function*/
declare total_sal number;
begin
  total_sal := total_salary(2);
  DBMS_OUTPUT.PUT_LINE('Total Salary is :|| total_sal);
end;
      
```
- Script Output Tab:** Located at the bottom, it shows the execution results:
 

```

Task completed in 0.004 seconds

anonymous block completed
Total Salary is :1950
      
```

iii. Create a function that calculates the maximum Salary.



The screenshot shows a SQL Developer window with a tab for 'user1\_conn'. The main editor displays a PL/SQL script. The script first creates a function named 'max\_salary' that takes no arguments and returns a number. It uses a 'select max(salary) into max\_sal from employees;' statement to find the maximum salary. Then, it declares a variable 'max\_sal' of type 'number' and executes the 'max\_salary' function, outputting the result using 'DBMS\_OUTPUT.PUT\_LINE'. The 'Script Output' window at the bottom shows the task completed in 0.003 seconds and the output 'Maximum Salary: 700'.

```
/*Create a function that calculates the maximum Salary*/  
create or replace function max_salary return number  
is max_sal number;  
begin  
    select max(salary)  
    into max_sal  
    from employees;  
    return max_sal;  
end;  
  
/* execute max_salary function */  
declare  
    max_sal number;  
begin  
    max_sal := max_salary();  
    DBMS_OUTPUT.PUT_LINE('Maximum Salary: ' || max_sal);  
end;
```

Script Output x  
Task completed in 0.003 seconds  
anonymous block completed  
Maximum Salary: 700

**Team Members :**

Students of the [Faculty of Computers and Artificial Intelligence](#) at [Helwan University](#), Department of Computer Science

Course : Database-II(fall 2023)

<b>Name</b>	<b>ID</b>
<b>Ossama Samir</b>	<b>20210140</b>
<b>Abd-ulla Mohamed</b>	<b>20210556</b>
<b>Marowa Omar</b>	<b>20210900</b>
<b>Rana Essam</b>	<b>20210335</b>
<b>Toka Mohamed</b>	<b>20210242</b>
<b>Abd-ulla Yasser</b>	<b>20210564</b>
<b>Omar Nasser</b>	<b>20210628</b>
<b>Seif El-den Samy</b>	<b>20210437</b>